

# Taller cuatro. Sobrecarga de constructores y métodos y destructores.

Programación orientada objetos – 2021-1S



Simón Cuartas Rendón – C.C. 1.037.670.103

Estudiante de estadística

## Parte uno. Ejercicio para analizar.

### A. ¿Qué pasaría si se pierde la referencia de un objeto tipo *Persona*?

En este caso ocurriría que, al ya no existir ninguna referencia a algún objeto de esta clase, se ejecutaría el método `finalize()` que está asociado a la clase *Persona*, de modo que se va a imprimir una línea que menciona “Matando a XXXX”, donde XXXX se refiere al atributo nombre del objeto de esta clase cuya referencia se ha perdido. Además, con esto, el recolector de basura del sistema operativo estará listo para poder eliminar de la memoria dinámica la información relacionada con este objeto.

### B. ¿Cómo podríamos conocer el nombre del dueño de la variable `auto` de la línea once del método `main`?

Para este caso habría que escribir:

```
auto.getDueno().getNombre();
```

con la finalidad de obtener únicamente el nombre de la persona. Aquí hay que tener presente que si se omite el `getNombre()`, entonces se va a ejecutar el `toString()` de la clase *Vehículo*, el cual antecedería un “Soy” antes del nombre, que no es lo que queremos particularmente en este caso.

```
<terminated> ObjTaller4 [Jav
Alexander
Matando a: Santiago
Soy Santiago
```

### C. ¿De qué manera podemos agregar un dueño al vehículo de la línea trece del método `main`?

Al definir la clase *Vehículo*, se implementó un setter para el atributo dueno, así que podemos echar mano de tal método para poder indicarle cuál va a ser su dueño. Así, suponiendo que se quiere que sea el objeto de la clase *Persona Daniel* el dueño de tal carro, entonces habría que añadir la siguiente línea de código:

```
auto2.setDueno(personas[2]);
```

Y al hacer esto, podemos implementar una línea adicional de código semejante a la anterior para saber verificar que se ha hecho bien.

```
System.out.println("Yo, " + auto2.getDueno().getNombre() + ", soy el dueño del segundo carro.");
```

De este modo, al ejecutar el código se obtiene lo siguiente:

```

1 package parteUno;
2
3 public class ObjTaller4 {
4     public static void main(String[] args) {
5         String[] nombres = {"Alejandro", "Jaime", "Daniel",
6                             "Santiago", "Alexander"};
7         Persona[] personas = new Persona[5];
8
9         for (int i = 0; i < nombres.length; i++) {
10            personas[i] = new Persona(nombres[i]);
11        }
12
13        Vehiculo auto = new Vehiculo("ABC-306", Motor.ElMotor);
14        System.out.println("Yo, " + auto.getDueno().getNombre() + ", soy el dueño del primer carro.");
15
16        Vehiculo auto2 = new Vehiculo("TXT-452", Motor.ElMotor);
17        auto2.setDueno(personas[2]);
18        System.out.println("Yo, " + auto2.getDueno().getNombre() + ", soy el dueño del segundo carro.");
19
20        personas[0] = null;
21        personas[1] = personas[2];
22        personas[4] = null;
23        personas[3].finalize();
24        System.out.println(personas[3]);
25    }
26 }
27
28
29

```

Console Output:

```

<terminated> ObjTaller4 [Java Application] C:\Users\simon\p2\pool\plugins\org
Yo, Alexander, soy el dueño del primer carro.
Yo, Daniel, soy el dueño del segundo carro.
Matando a: Santiago
Soy Santiago

```

- D. Usando la variable `auto2` de la línea trece del método `main`, obtenga el valor del atributo `velocidadMaxima` del motor del vehículo. Adjunte su propuesta.

La idea en este caso es acceder a los atributos del objetos `auto2`, en particular del motor, para lo cual usamos `getMotor()`, y hecho esto, podemos acceder a los atributos del objeto de la clase `Motor` asociado al `auto2`, en particular a la velocidad máxima, para lo cual se emplea el método `getVelocidadMaxima()`, y nos quedaría toda la siguiente línea:

```
auto2.getMotor().getVelocidadMaxima()
```

Y si se le agrega un `System.out.println(...)`, podemos ver tal valor en consola, como se ve a continuación:

```

1 package parteUno;
2
3 public class ObjTaller4 {
4     public static void main(String[] args) {
5         String[] nombres = {"Alejandro", "Jaime", "Daniel",
6                             "Santiago", "Alexander"};
7         Persona[] personas = new Persona[5];
8
9         for (int i = 0; i < nombres.length; i++) {
10            personas[i] = new Persona(nombres[i]);
11        }
12
13        Vehiculo auto = new Vehiculo("ABC-306", Motor.ElMotor);
14        System.out.println("Yo, " + auto.getDueno().getNombre() + ", soy el dueño del primer carro.");
15
16        Vehiculo auto2 = new Vehiculo("TXT-452", Motor.ElMotor);
17        auto2.setDueno(personas[2]);
18        System.out.println("Yo, " + auto2.getDueno().getNombre() + ", soy el dueño del segundo carro.");
19
20        // Obtener la velocidad máxima del auto2.
21        System.out.println(auto2.getMotor().getVelocidadMaxima());
22    }
23 }

```

Console Output:

```

<terminated> ObjTaller4 [Java Application] C:\Users\simon\
Yo, Alexander, soy el dueño del primer carro.
Yo, Daniel, soy el dueño del segundo carro.
90
Matando a: Santiago
Soy Santiago

```

- E. Suponga que, al perder la referencia del objeto, se borra del sistema, es decir, el *garbage collector* es muy eficiente. ¿Qué imprimiría por consola lo anterior?

En este caso, el *garbage collector* va a borrar de la memoria todo aquello que pierda su referencia y se ejecutará entonces el método `finalize()`, así, se vería en consola algo de este estilo:

Matando a Alejandro

Matando a Jaime

Matando a Santiago.

Soy Santiago.

En el primer caso porque en la línea quince se está borrando la referencia de `personas[0]` a `Alejandro`, en el segundo caso porque se le está indicando a `personas[1]` que deje de apuntar a `Jaime` para hacerlo a `Daniel`, lo cual implica que el objeto `Jaime` pierde su apuntador, por lo cual se ejecuta el método `finalize()`, en el tercer caso porque se le está entregando a este último método mencionado el parámetro `persona[3]`, pero observar que en la última línea igual sale “Soy Santiago” dado que el método `finalize()` no elimina apuntadores o referencias como tal, sino que se ejecuta cuando estas dejan de existir o cuando es invocado directamente como en esta situación.

- F. ¿Qué ocurre al ejecutar la línea `System.out.println(personas[1])`?

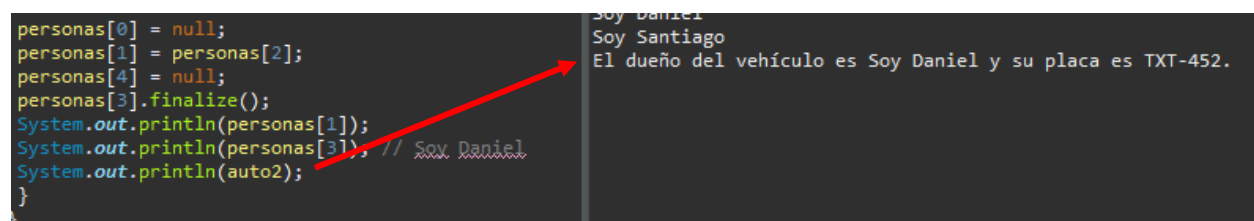
La respuesta depende de dónde ubiquemos tal sentencia en el método `main`, pues si lo escribimos justo antes de que se realice el ciclo `for`, por ejemplo, obtendremos un **error** en tanto aún no se ha declarado este objeto. Si lo hacemos luego del ciclo `for`, se obtendrá un “Soy Jaime”, pues este es el miembro con índice uno de la lista `personas[]`, y gracias al método `toString()` que tiene la clase `Personas` se obtiene el “Soy (...)” que sale al comienzo, pero si se realiza luego de la línea dieciséis, en la que se declara `personas[1] = personas[2]`, lo que va a suceder es que el apuntador de `personas[1]` va a dejar de lado al objeto `Jaime` para comenzar a apuntar al objeto `Daniel`, lo que resultará en un “Soy Daniel”.

- G. ¿Qué modificación se le debe hacer al código para que al momento de ejecutar `System.out.println(auto2)` me aparezca la placa del vehículo y el dueño de este?

Para conseguir obtener el dueño y la placa de `auto2`, y general, para cualquier objeto de la clase `Vehículo` que se pase como parámetro de un `System.out.println(...)` basta con definir un método `toString()` en la clase `Vehículo` que retorne tal resultado deseado. A continuación se muestra esta implementación:

```
public String toString() {  
    return "El dueño del vehículo es " + dueño + " y su placa es " + placa + ".";  
}
```

Y el resultado de esto es:



```
personas[0] = null;  
personas[1] = personas[2];  
personas[4] = null;  
personas[3].finalize();  
System.out.println(personas[1]);  
System.out.println(personas[3]); // Soy Daniel  
System.out.println(auto2);  
}
```

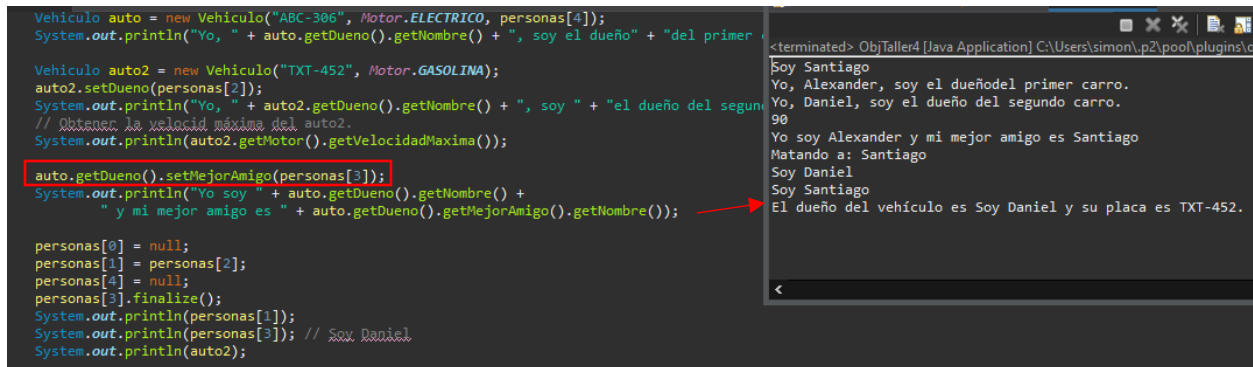
Soy Daniel  
Soy Santiago  
El dueño del vehículo es Soy Daniel y su placa es TXT-452.

- H. Usando la variable `auto` de la línea once del método `main` y usando el atributo `dueno_1` asignar al tercer elemento de la lista `Personas` como mejor amigo. Adjuntar propuesta.

El código agregado para poder realizar esta asignación es:

```
auto.getDueno().setMejorAmigo(personas[3]); // << Línea principal para este punto.
System.out.println("Yo soy " + auto.getDueno().getNombre() +
    " y mi mejor amigo es " + auto.getDueno().getMejorAmigo().getNombre());
```

Y el resultado en pantalla de de esto es:



```
Vehiculo auto = new Vehiculo("ABC-306", Motor.ELECTRICO, personas[4]);
System.out.println("Yo, " + auto.getDueno().getNombre() + ", soy el dueño " + "del primer ");

Vehiculo auto2 = new Vehiculo("TXT-452", Motor.GASOLINA);
auto2.setDueno(personas[2]);
System.out.println("Yo, " + auto2.getDueno().getNombre() + ", soy " + "el dueño del segun");
// Obtener la velocidad máxima del auto2.
System.out.println(auto2.getMotor().getVelocidadMaxima());

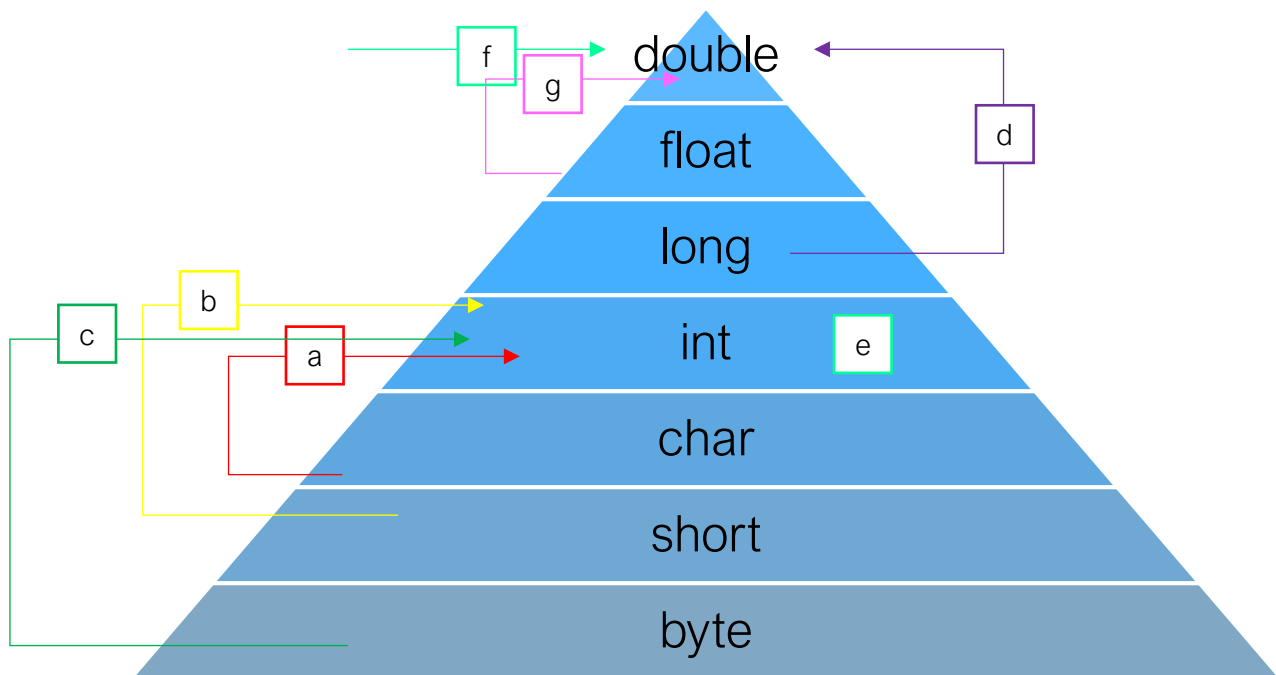
auto.getDueno().setMejorAmigo(personas[3]);
System.out.println("Yo soy " + auto.getDueno().getNombre() +
    " y mi mejor amigo es " + auto.getDueno().getMejorAmigo().getNombre());

personas[0] = null;
personas[1] = personas[2];
personas[4] = null;
personas[3].finalize();
System.out.println(personas[1]);
System.out.println(personas[3]); // Soy Daniel.
System.out.println(auto2);
```

```
<terminated> ObjTaller4 [Java Application] C:\Users\simon\p2\pool\plugins\o
Soy Santiago
Yo, Alexander, soy el dueño del primer carro.
Yo, Daniel, soy el dueño del segundo carro.
90
Yo soy Alexander y mi mejor amigo es Santiago
Matando a: Santiago
Soy Daniel
Soy Santiago
El dueño del vehículo es Soy Daniel y su placa es TXT-452.
```

## Parte dos. Ejercicio para analizar.

- A. Según el siguiente código, indique qué se imprime en cada línea y explique el porqué de cada línea donde se imprime.
- Línea 13.** Se ejecuta en el método que recibe un `int`, ya que el tipo de dato primitivo más cercano y que está por encima en la 'jerarquía' de datos primitivos. Aquí, se observa que se devuelve un valor numérico, el cual corresponde al valor **ASCII** del carácter 'c'.
  - Línea 14.** Para esta situación, se tiene un valor de tipo `short`, el cual entra al método que recibe tipo `int`, y al ser un valor numérico, no tiene la necesidad de cambiar su valor como en el caso anterior.
  - Línea 15.** Semejante a los casos anterior, el `byte` tiene al tipo primitivo `int` como el tipo superior más cercano, por lo que la función ejecutada es la misma de los tres casos anteriores, y al consistir en un valor numérico, sale tal cual por consola.
  - Línea 16.** Aquí se le está entrando como parámetro a la función un valor de tipo `long`, el cual es superior al tipo `int`, de modo que busca al superior más cercano que para su caso es el `double`, y así, se imprime en forma de notación científica.
  - Línea 17.** Como se trata de un `int`, se imprime directo a través de la función que recibe como parámetro un `int`.
  - Línea 18.** Como se trata de un `double`, se imprime directo a través de la función que se recibe como parámetro un `double`.
  - Línea 19.** En este caso se tiene un `float` que entonces ingresa al `double` al ser el tipo primitivo más próximo por encima. Además, se observa que no se obtiene exactamente el mismo valor que se ingresó originalmente, sino uno con una modificación en su valor decimal, lo cual es consecuencia del sistema binario del computador por el cual se pierde precisión.



- B. Realizar los cambios que se indican a continuación y explicar cómo se obtiene el resultado y por qué cambia respecto al punto anterior.
- Activar la función que recibe un short.** En este caso cambian los resultados de la ejecución de las líneas catorce y quince, pues inicialmente debían ir hasta la función que recibe un **int** al ser el único tipo primitivo por encima con el cual podían ejecutarse, pero al activar la opción del **short**, estos ingresan por este: para la línea catorce porque este mismo es un **short** y para la línea quince porque para el **byte** el **short** es el tipo primitivo superior más cercano.
  - Activar la opción que recibe un float.** En este caso, los casos de las líneas dieciséis y diecinueve ya no tendrán la necesidad de 'ir' hasta un **double** sino que el mismo **float** se convierte en el tipo primitivo más cercano: en el caso de la línea dieciséis porque **long** está justo por debajo de **float** y en este caso lo redondea a  $10^9$  al estar usando notación científica, mientras que con la línea diecisiete, al ser del mismo tipo, se ingresa por ahí mismo.
  - Activar la opción que recibe un float y desactivar la que recibe un double.** En este caso el programa no va a ser capaz de ejecutarse ya que estamos solicitando que corra la función **function()** con un **double** como parámetro, pero no existe ningún método que lo pueda correr, por lo que se debe hacer un **cast** explícito para que el programa se pueda ejecutar.
  - Comentar todas las funciones excepto la que tiene un double.** En este caso, al ser el **double** el tipo primitivo 'mayor', entonces todas las variables se pueden ejecutar en esta mediante un **cast implícito**, considerando que al ser ahora tomadas como de tipo **double**, entonces serán consideradas decimales (por lo que se les agrega el punto cero (.0) cuando son del tipo que recibe números enteros) y hace las aproximaciones computacionales correspondientes en el caso del **float** con pérdida de precisión al ser el computador programado en sistema binario.

## Parte tres. Git-Hub.

1. **¿Qué utilidad tiene la sobrecarga de métodos? ¿Por qué no simplemente llamarlos con distintos nombres?**

La sobrecarga permite ejecutar una función con características semejantes a la original pero con una cantidad de parámetros diferentes, es decir, con una variación en su firma, lo cual permite al programador y al usuario dado el caso utilizar una misma función pero con una variación en sus parámetros, lo cual resulta útil en cuanto a la nemotecnia en el sentido que no se van a tener que registrar múltiples funciones que tal vez se olvidarán.

2. **¿Por qué cree que la firma de un método no está compuesta por el nombre de los argumentos ni el tipo de retorno para diferenciarlo de los demás?**

Porque el objetivo de la firma es justamente diferenciarla a través del nombre para diferenciarla en primera instancia de otros métodos y de acuerdo con los parámetros en el caso de métodos sobrecargados. En este sentido, es posible que el diseñador desee que una función que recibe unos parámetros particulares tenga una visibilidad o un retorno diferente de acuerdo con los parámetros que se le están dando, por lo cual sería torpe incorporar estos aspectos a la firma de un método. Además, si se desea que un par de métodos con igual denominación y mismas entradas tengan un retorno diferente, puede ser conveniente darles nombres diferentes para poder distinguirlos claramente entre sí.

3. **¿Qué ventajas puede encontrar en un inicializador estático frente a simplemente definir el valor inicial del atributo en la declaración de este?**

Podría ser útil para reunir a todas las variables o atributos estáticos en un solo bloque y lograr inicializarlos de manera inmediata, además de aislarlos de los atributos de instancia, lo cual permite identificarlos rápidamente y, además, se logra un ahorro de escritura de la palabra `static` al tener que hacerlo solo una vez.

4. **¿Qué ventajas tiene poder llamar a otro constructor con la expresión `this`?**

Se ahorra escritura en el sentido que no hay que repetir los parámetros, sino que solo se modifican los de interés y se le sitúan algunos parámetros por defectos a los que no están relacionados con el constructor específico que se está definiendo.